НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Обчислення в обчислювальних графах: зворотне поширення помилки

Дмитро Пінчук

14 вересня 2025 р.

1 Обчислення в обчислювальних графах: зворотне поширення помилки

1.1 Вступ

З самого початку історії, людство намагається копіювати вже існуючі в природі речі для вирішення своїх проблем і нейронні мережі не є виключенням. Бажання описати процеси мислення в людському мозку спочатку було лише справою стародавніх філософів, але з часом "перекочувало"в математику, де мозок намагались описати точною математичною моделлю, та зі зростанням зацікавленості цією темою, вона взагалі переросла в окрему наукову сферу - машинне навчання.

Завдяки розвитку комп'ютерної сфери, коли кожному став доступним персональний комп'ютер, відбувся різкий стрибок у вивченні цієї теми тому, що науковці змогли перейти від теоретичних розрахунків на папері до практики і значно полегшити та прискорити ці розрахунки за допомогою програм. Сьогодні здобутками науковців та дослідників цієї сфери користуються мільйони людей по всьому світу для найрізноманітніших цілей - від прогнозів майбутнього до персональних асистентів та творчості, і з кожним роком важче уявити повсякденне життя без використання нейронних мереж [1, 2].

Сучасні нейронні мережі, якими ми користуємось сьогодні, за своєю технічною складністю в десятки разів перевершують ті моделі, які науковці досліджували за останні 50 років - від простих одношарових мереж та перцептронів ми перейшли до згорткових, генеративних та рекурентних мереж. Важливу роль в їх архітектурі відображає метод зворотного поширення помилки через те, що він безпосередньо впливає на процес навчання та його якість. Звісно ж, що зі збільшенням складності зросла не тільких кількість розрахунків, а й системні вимоги до комп'ютерів, які їх проводять.

Оскільки нейронні мережі являють собою складні та великі за розміром математичні моделі, то для подальших досліджень виникла необхідність у такому способі їх зображення який був би доволі компактним, але при цьому максимально інформативним та наглядно показував зв'зяки між різними частинами моделі. Таке рішення було знайдено в іншому розділі математики (теорія графів) та отримало назву Обчислювальний граф і через його зручність досі використовується в сучасній літературі. В наступному розділі буде приділено увагу саме цьому способу зображення обчислень [3, 4].

1.2 Основи обчислювальних графів

Обчислювальний граф - це орієнтований граф, який використовується для представлення обчислень та математичних виразів. Вхідні дані, змінні, а також вихідні дані зображають як вершини цього графа, а ребра відповідно показують напрямок потоку даних між вершинами (частинами системи). Завдяки цьому, такий граф дозволяє явно описати структуру обчислень, що спрощує їх реалізацію та оптимізацію. Кожній функції відповідає елементарна операція, яка позначається вершиною графа. Складні функції розбиваються на декілька примітивних операцій, поєднаних між собою [4].

Розглянемо будову графу на прикладі простого математичного виразу:

$$Y = (a + b) * (b - c), \tag{1}$$

де a, b, c - довільні раціональні числа.

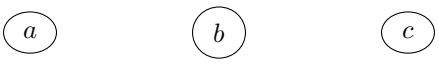
Даний вираз складається з 3 простих операцій, тому розділимо його на окремі операції, враховуючи їх пріоритет та запишемо кожен підвираз із власною назвою змінної:

$$d = a + b \tag{2}$$

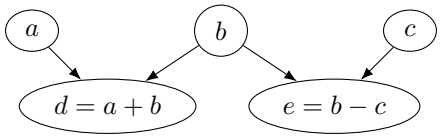
$$e = b - c \tag{3}$$

$$Y = d * e \tag{4}$$

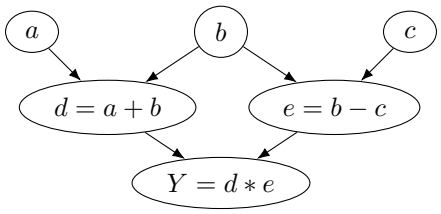
Таким чином, маємо 3 вхідні змінні та 2 проміжні. Так як вираз вже спрощений до примітивних операцій (додавання, множення та віднімання) то можемо перейти до побудови зображення графу, який в нашому випадку зображено згори донизу. Побудуємо вершини для вхідних змінних a, b та c:



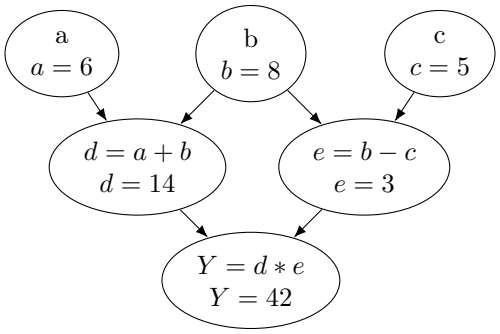
Тепер додаємо вершини для проміжних змінних d і e та з'єднаємо їх ребрами:



Залишилось лише поєднати розрахунки проміжних змінних d і e у вихідну вершину Y. Готовий обчислювальний граф виглядає наступним чином:



Для наочності спробуємо підставити довільні числа замість змінних і провести розрахунки. Нехай, $a = 6$, $b = 8$, $c = 5$. Тоді наш граф буде виглядати наступним чином:



З нього видно, що результат виконання виразу $Y = 42$. Отже, для такого невеликого виразу подібний метод зображення може виглядати занадто громіздким, але він є набагато ефективнішим, коли кількість вхідних та проміжних змінних переходить межу в сотні та тисячі штук. Цей спосіб зображення використовується і в тому числі при поясненні принципу роботи зворотного поширення помилки у великих багатошарових нейронних мережах. Зазвичай, в таких випадках подібні змінні групують між собою для того, а також багато однакових прихованих шарів опускають щоб обчислювальний граф мав адекватні розміри [4, 5].

В наступному розділі буде коротко розглянуто історію виникнення методу зворотного поширення помилки, після чого дізнаємось принцип його роботи та роль, яку він відіграє в архітектурі нейронної мережі.

1.3 Історія виникнення методу зворотного поширення помилки

Перші методи для тренування нейронних мереж не використовували зворотне поширення помилки. Так, спочатку в 1950-х роках Розенблат представив свій алгоритм навчання для одношарового перцепторна. На той час це був новий клас нейронних мереж, але досить швидко в ньому знайшли істотні недоліки, які обмежували його застосування для широкого спектру задач. Наприклад, одношаровий перцпетрон не міг реалізувати деякі елементарні функції (логічна функція XOR). Було випущено книгу "Перцептрони"авторства Марвіна Мінського та Сеймура Паперта, де вони продемонстрували ці обмеження. Стала зрозумілою необхідність в багатошаровому перцептроні з прихованими внутрішніми шарами нейронів, що допомогло вирішити деякі фундаментальні недоліки.

Після цього в 1965 році Олексієм Івахненко було запропоновано алгоритм прямого поширення для навчання багатошарових перцептронів під назвою "метод групового урахування аргументів". Такі нейронні мережі прийнято вважати першими "глибокими які могли навчатися створювати розподілені та ієрархічні внутрішні зв'язки для представлення даних.

Згодом, у 1970 році, коли була опублікована магістерська дисертація за авторства Сеппо Ліннайнмаа, фактично покладено початок методу, який відомий нам під сучасною назвою "метод зворотного поширення помилки"(backpropagation), або ж "зворотний режим автоматичного диференціювання"(reverse mode of automatic differentiation).

Цікавим фактом є те, що алгоритм в його роботі представлено в більш узагальненому вигляді і він не був напряму асоційованим з нейронними мережами. Крім того, до його праці було додано реалізацію цього алгоритму, написану мовою програмування Fortran. В майбутньому саме цей метод стане основою для реалізації сучасних нейронних мереж і широко розповсюдиться в популярних інструментах для створення нейронних мереж, таких як Tensorflow.

Незважаючи на першопроходця Сеппо, цей алгоритм був фактично заново винайдений декількома іншими дослідниками протягом деякого часу. Тут важливо відзначити роботи Пола Вербоса (1982) та Девіда Румельхарта та інших (1986), після яких зворотне поширення фактично стало стандартом для навчання нейронних мереж [6, 7, 8, 9, 10].

Перейдемо до безпосереднього принципу роботи цього метода в наступному розділі.

1.4 Принцип роботи зворотного поширення помилки

На даному рисунку схематично зображено приклад одного з видів нейронних мереж - багатошарового перцептрона за допомогою обчислювального графа.

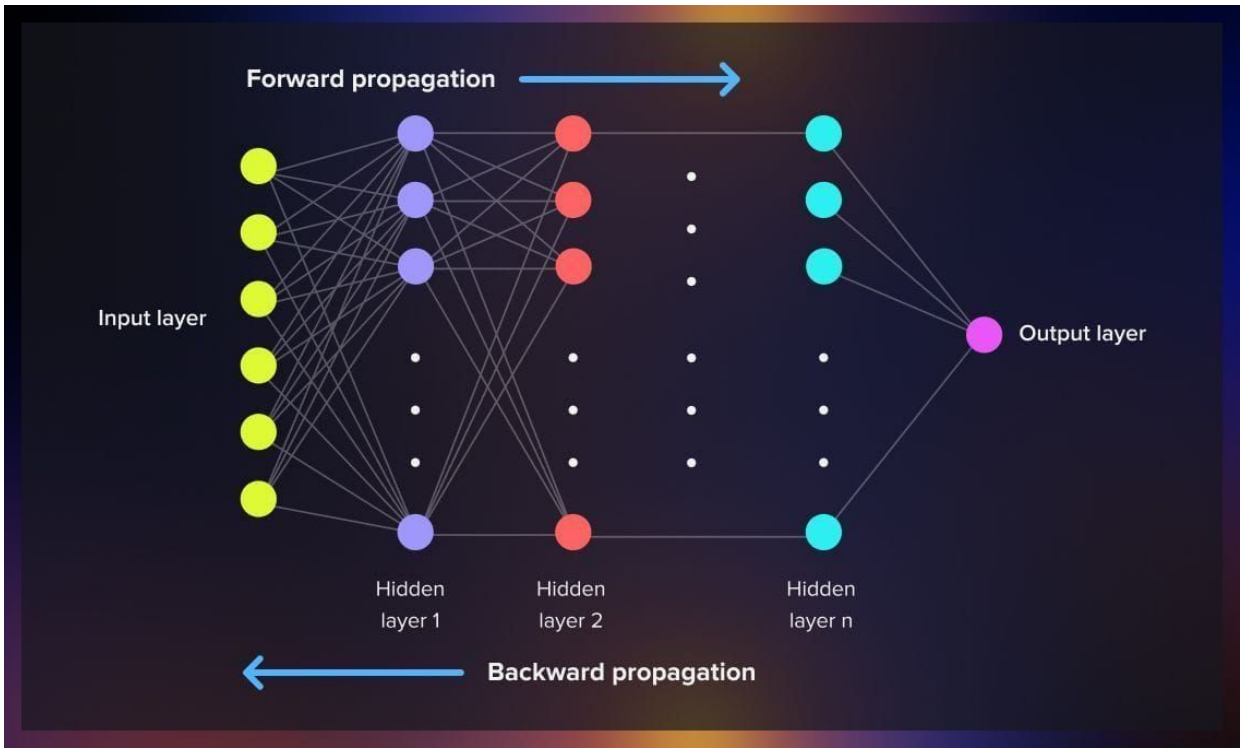


Рис. 1: Пряме та зворотне поширення в типовому багатошаровому перцептроні.

Як можна побачити на рисунку, дані проходять від вхідного шару через приховані шари до вихідних і ці з'єднання є двонаправленими. В більшості сучасних нейронних мереж одночасно застосовують як пряме, так і зворотне поширення, тому перед тим як розглядати зворотне поширення, варто дізнатись про його предка - пряме поширення.

Пряме поширення являє собою процес обчислення значень у вершинах обчислювального графа в топологічному порядку, починаючи з вхідних даних і закінчуючи вихідними результатами. Кожна вершина графа обчислюється шляхом застосування відповідної операції над значеннями, що надходять від попередніх вершин (вхідні дані або результати проміжних операцій). У контексті нейронних мереж цей процес відповідає передачі сигналів від вхідного шару до вихідного шару з урахуванням ваги кожної вершини графа.

Розглянемо як виглядає цей процес з математичної точки зору. У процесі прямого поширення вхідний сигнал x_i проходить через шари нейронної мережі, генеруючи вихідний сигнал y. Для кожного шару обчислення відбувається за формулою:

$$y = w_1 x_1 + ... + w_d x_d + b, \tag{5}$$

де w_d - значення ваги вершини, x_d - значення вхідного сигналу, b - значення зсуву.

Для спрощення, щоб не розписувати кожну пару ваги-значення (яких потенційно може бути тисячі), використовується така форма запису із застосуванням векторного добутку:

$$y = \mathbf{W}^\top \mathbf{X} + b, \tag{6}$$

де W - вектор ваги нейронів, X - вектор вхідних значень, b - значення зсуву.

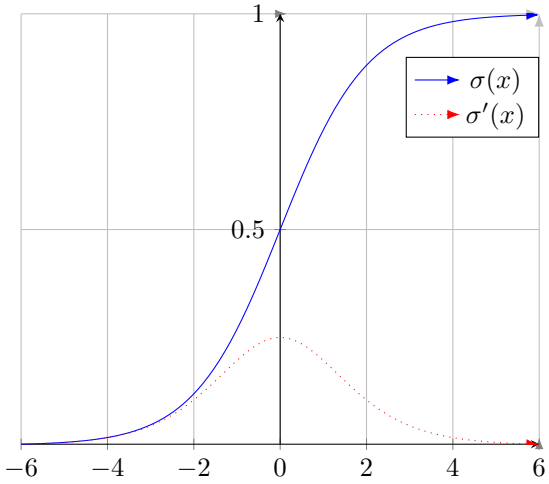
Відповідно, тепер отримане значення необхідно пропустити через функцію активації:

$$\hat{y} = \sigma(y) \tag{7}$$

Наприклад, можна взяти сигмоїду в якості функції активації:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

Тоді, графік для цієї функції активації та її похідної буде виглядати таким чином:



Таким чином, ми отримаємо вектор зі значеннями виходів, які й будуть результатом роботи мережі.

На цьому перший етап - пряме поширення завершено і далі можна переходити до розгляду алгоритму зворотного поширення помилки [9, 11, 12, 13].

Метод зворотного поширення помилки - це ітеративний градієнтний алгоритм, який застосовують для мінімізації помилки роботи багатошарової мережі з метою корекції прогнозу для отримання бажаного виходу. Він фактично являє собою похідну значення прямого поширення. Основна його ідея полягає у поширенні сигналів помилки від виходів мережі до її входів, у напрямку, оберненому до прямого поширення сигналів у нормальному режимі роботи. Він також має альтернативну назву "зворотний режим автоматичного диференціювання"в інших сферах, не пов'язаних з нейронними мережами.

Використовуючи результати попередніх розрахунків для прямого поширення, необхідно розрахувати функцію втрат, опираючись на початковий датасет для тренування мережі. Функція втрат використовується для подальшої корекції роботи нейронної мережі за допомогою градієнтного спуску. Для

різних видів мереж ця функція буде залежати від функції активації та інших факторів. Наприклад, для задачі бінарної класифікації використовується перехресна ентропія, яка розраховується за наступною формулою:

$$J(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}\Big[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\Big], \tag{9}$$

де $\hat{y}$ - це вихідні значення нейронів після застосування функції активації, а y - початкові значення з тренувального датасета. На виході ми повинні отримати новий вектор такої ж розмірності, як і вихідний шар нейронної мережі.

Наступним кроком необхідно розрахувати градієнти цільової функції. Для цього цільову функцію (функцію втрат) розбивають на часткові похідні, які розраховуються окремо за наступними формулами:

$$\begin{aligned}
\frac{\partial J(\hat{y}, y)}{\partial \hat{y}} &= \frac{1}{n}\big[-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\big] \\
\frac{\partial J(\hat{y}, y)}{\partial z} &= \frac{\partial J(\hat{y}, y)}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z} = \frac{1}{n}\big(\hat{y} - y\big) \\
\frac{\partial J(\hat{y}, y)}{\partial W} &= \frac{\partial J(\hat{y}, y)}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial W} = \frac{1}{n}(\hat{y} - y)\cdot X^{\intercal} \\
\frac{\partial J(\hat{y}, y)}{\partial b} &= \frac{\partial J(\hat{y}, y)}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\big(\hat{y}^{(i)} - y^{(i)}\big)
\end{aligned} \tag{10}$$

Після того, як градієнти функції пораховані, ми фактично маємо коефіцієнти, які допоможуть підкорегувати існуючі ваги нейронів в усіх шарах, що дозволить збільшити кількість правильних результатів при наступній ітерації. Для цього необхідно взяти вектор градієнтів і за допомогою нього оновити ваги нейронів та зсув за наступною формулою:

$$\begin{aligned}
W &= W - \alpha\frac{\partial J(\hat{y}, y)}{\partial W} \\
b &= b - \alpha\frac{\partial J(\hat{y}, y)}{\partial b},
\end{aligned} \tag{11}$$

де W - вектор ваги нейронів; α - швидкість навчання (крок); ∂J - часткова похідна функції втрат; $\hat{y}$ - вихідні значення нейронів; y - тренувальні значення з датасету; ∂W - часткова похідна вектора ваги нейронів; b - значення зсуву; ∂b - часткова похідна значення зсуву.

Всі вищезазначені кроки повторюються для кожної ітерації навчання з підстановкою необхідних змінних. Але для оптимізації процесу навчання зворотне поширення запускають лише для деякого відсотку всіх ітерацій (наприклад, лише для кожної 20-ї ітерації) з усередненими значеннями за всі ці ітерації, оскільки це допомагає значно пришвидшити процес навчання мережі та знизити використання ресурсів системи [12, 11, 9, 14, 15].

1.5 Висновки

У даній роботі було розглянуто принцип роботи методу зворотного поширення помилки з використанням обчислювальних графів. Мною було описано теоретичні основи обчислювальних графів та наведено приклад для простого виразу.

Також було проведено історичний аналіз виникнення методу зворотного поширення. Історичний аналіз показує, що розвиток цього методу не був одиничним проривом, а скоріше еволюційним процесом зі значними внесками багатьох дослідників протягом останніх 50 років. Починаючи з початкової роботи Сеппо Ліннайнмаа у 1970 році, через дослідження Пола Вербоса у 1982 році, і завершуючи широко визнаною реалізацією Румельхарта у 1986 році, цей метод поступово став основою для навчання сучасних нейронних мереж.

Окрім цього, було приведено математичний апарат, який стоїть у основі алгоритму і за допомогою якого цей метод ефективно оптимізує роботу нейронної мережі. Усі необхідні кроки виконуються в два етапи: пряме поширення для прогнозування та зворотне поширення для корекції помилок. Цей процес у поєднанні з оптимізацією градієнтного спуску дозволяє нейронним мережам навчатися та ітеративно покращувати свою точність.

Метод зворотного поширення помилки, незважаючи на свою тривалу історію продовжує відігравати центральну роль у розвитку штучного інтелекту та має значні перспективи для подальшого розвитку та вдосконалення.

У короткостроковій перспективі очікується подальша оптимізація методу для роботи з масштабними моделями. Зростаюча потужність обчислювальних систем та поява спеціалізованих процесорів для машинного навчання відкривають можливості для впровадження більш складних варіацій алгоритму. Особливо перспективним є розвиток методів паралельного обчислення градієнтів, що дозволить значно прискорити процес навчання великих нейронних мереж.

У довгостроковій перспективі він стане основою для розробки більш енергоефективних алгоритмів навчання, що є актуальним на сьогодні. Дослідники також працюють над адаптацією методу для нових архітектур нейронних мереж, включаючи квантові обчислення та нейроморфні системи [16, 17].

Важливо відзначити, що хоча з'являються нові методи навчання нейронних мереж, базові принципи зворотного поширення помилки залишаються незмінними. Цей метод продовжує вдосконалюватися та адаптуватися до нових викликів, залишаючись одним зі стовпів сучасного машинного навчання.

Література

[1] P. Ekwere. (2024) The history of artificial intelligence: From ancient myths to modern machines, from turing to tomorrow. AI Accelerator Institute. [Online]. Available: https://www.aiacceleratorinstitute.com/ai-ancient-myths-to-modern-machines-turing-to-tomorrow/

[2] AIPRM. (2024) Ai statistics 2024 - aiprm. [Online]. Available: https://www.aiprm.com/ai-statistics/

[3] (2024) 8 common types of neural networks. Coursera. [Online]. Available: https://www.coursera.org/in/articles/types-of-neural-networks/

[4] (2023) Computational graphs in deep learning. GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/computational-graphs-in-deep-learning/

[5] Y. Artzi. (2017) Computation graphs. [Online]. Available: https://www.cs.cornell.edu/courses/cs5740/2017sp/lectures/04-nn-compgraph.pdf

[6] J. Schmidhuber. (2014) Who invented backpropagation? [Online]. Available: https://people.idsia.ch/~juergen/who-invented-backpropagation.html

[7] C. Olah. (2015) Calculus on computational graphs: Backpropagation – colah's blog. [Online]. Available: https://colah.github.io/posts/2015-08-Backprop/

[8] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors," *Master's Thesis (in Finnish), University of Helsinki*, pp. 6–7, 1970.

[9] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2020, https://d2l.ai.

[10] . R. J. W. David E. Rumelhart, Geoffrey E. Hinton. (1986) Learning representations by back-propagating errors. [Online]. Available: http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf

[11] G. Sanderson. Neural networks. 3Blue1Brown. [Online]. Available: https://www.3blue1brown.com/topics/neural-networks

[12] I. Logunova. (2023) Backpropagation in neural networks. [Online]. Available: https://serokell.io/blog/understanding-backpropagation

[13] Д.О. (2019) Дослідження нейромережевих методів аналізу медичних зображень. [Online]. Available: https://openarchive.nure.ua/server/api/core/bitstreams/99024a3f-c516-4f8f-84a9-908f54de8598/content

[14] W. contributors. (2024) Cross-entropy. [Online]. Available: https://en.wikipedia.org/wiki/Cross-entropy

[15] Y. Kochura. (2023) Ykochura ai-lab Логістична регресія. [Online]. Available: https://github.com/YKochura/ai-lab/blob/main/logistic-regression/logistic_regression.ipynb

[16] S. R. Gundu. (2024) Quantum neural networks: A novel architecture for hybrid quantum-classical algorithms. [Online]. Available: https://www.researchsquare.com/article/rs-3951691/v1

[17] Z. B. Houidi. (2024) Backpropagation-based recollection of memories: Biological plausibility and computational efficiency. [Online]. Available: https://www.biorxiv.org/content/10.1101/2024.02.05.578854v1.full